兰州大学

本科毕业论文

论文题目（中文） 基于 Deep Koopman 算子网络的非线性系统强化学习研究

论文题目（英文） Deep Koopman Network Based Reinforcement Learning of Nonlinear System

学生姓名 许恣欢

指导教师 赵东东

学　　院 信息科学与工程学院

专　　业 电子信息科学与工程

年　　级 2020 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人、集体已经发表或未发表的论文。

本声明的法律责任由本人承担。

论文作者签名：签名　　日　　期：签名

关于毕业论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文（设计）。本人离校后发表、使用毕业论文（设计）或与该毕业论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文（设计）研究内容：

☑可以公开

□不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名：签名　　导师签名：签名

日　　期：签名　　日　　期：签名

基于 Deep Koopman 算子网络的非线性系统强化学习研究

中文摘要

我的摘要

关键词： Koopman 算子理论，深度神经网络，强化学习

Deep Koopman Network Based Reinforcement Learning of Nonlinear System

Abstract

My Abstract

Keywords: Koopman Operator Theory, Deep Neural Network, Reinforcement Learning

目 录

第一章 绪 论

这是我的绪论 [1]

第二章 背景知识

在本章中，首先讨论一下有关的背景理论与算法。介绍一下 Koopman 算子理论（Koopman Operator Theory），并讨论 Koopman 算子对于重塑强化学习（Reinforcement Learning）中使用的马尔可夫决策过程（Markov Decision Process）的重要作用。同时，对于 Koopman 算子理论与深度神经网络（Deep Neural Network）之间的关联。

2.1 Koopman 算子理论

系统的强非线性是数据驱动建模和控制领域的核心问题之一，包括基于现代强化学习框架所做的工作。Koopman 算子理论 [cite] 为上述问题提出了一种解决方案。在该理论中，非线性系统动力学可以在提升到无限维度希尔伯特空间时变为线性系统，并可以通过一组新的基底对于该线性系统进行观测。升维工作已被证明对于线性化和简化某些具有挑战性的问题具有显著效果，这与机器学习领域中其他的类似努力相一致。

2.1.1 Koopman 算子与非线性系统演化

我们应该采用不同的形式对不同类型的系统中系统状态 x 进行描述。假设系统为确定性自治系统，我可以采用下面的方式，将一个离散动力系统描述为

$$\dot{x} = F(x) \tag{1}$$

其中，$\dot{x}$ 表示下一个时刻的系统状态；或者，采取不同的方式，将一个连续动力系统描述为

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t)) \tag{2}$$

而在处理实际问题是，我们通常考虑的是离散动力系统。所以我们就可以将连续系统通过流映射算子（Flow Map Operator）归纳为一个离散系统，系统演化如下

$$x(t+\tau) = F_\tau(x(t)) = x(t) + \int_t^{t+\tau} f(x(s))\mathrm{d}s \tag{3}$$

Koopman 算子提供了解决非线性系统控制问题的一个新的着眼点。在形式上，我们考虑一个实值向量测量函数 $g : M \to \mathbb{R}$，且都由无限维希尔伯特空间的元素组成，其中 M 是一个流形。通常，这个流形被认为是 $L^\infty(X)$, $X \subset \mathbb{R}^d$。一般情况下，函数 g 被称为可观测函数。Koopman 算子理论中指出，Koopman 算子 $\mathcal{K}$ 和 Koopman 生成器 $\mathcal{L}$ 都是作用于上述观测函数 g 的无限维线性算子，在确定性系统中，有

$$\mathcal{K}g = g \circ F \tag{4a}$$

$$\mathcal{L}g = f \cdot \nabla g \tag{4b}$$

Koopman 生成器 $\mathcal{L}$ 和 Koopman 算子有如下的关系

$$\mathcal{L}g = \lim_{t\to 0}\frac{\mathcal{K}g - g}{t} = \lim_{t\to 0}\frac{g\circ F - g}{t} \tag{5}$$

Koopman 算子理论可以更广泛地应用于任何马尔可夫过程，但本文以随机性连续时间系统为例，此时，Koopman 算子的定义如下:

$$\mathcal{K}g = \mathbb{E}(g(X)|X_0 = \cdot)$$
$$\mathcal{L}g = \lim_{t\to 0}\frac{\mathcal{K}g - g}{t}$$

Koopman 算子将测量函数 g 沿着路径 x 向前演化如下：

$$\mathcal{K}_\tau g(x_t) := g(F_\tau(x_t)) = g(x_{t+\tau}) \tag{6}$$

其中 F 代表着系统的演化规律。更一般的, 在随机自治系统中，$\mathcal{K}$ 被如下定义为条件预测算子：

$$\mathcal{K}g(x_t) = \mathbb{E}\left[g(X_{t+\tau})|X_t = x_t\right] \tag{7}$$

在上述离散系统算子中，普遍使用 $\mathcal{K} := \mathcal{K}_1$，在本文中也照此用法。

2.1.2 Koopman 算子本征函数

上文提出，可观测函数 g 的观测，都存在于无限维的希尔伯特空间中（被称作观测空间），被如 Koopman 算子 $\mathcal{K}$ 等无限维的算子推动沿着给定非线性动力学系统演化。因此不难发现，我们可以应用 Koopman 算子理论研究，将对于非线性系统的研究，通过观测空间状态线性演化实现。

同时，由于难以捕捉无限维希尔伯特空间中所有可观测函数的演化，所以应该试图识别随着非线性系统动力学而线性演化的关键观测函数，Koopman 算子的本征函数就可以作为一组特殊的观测函数 [cite]：

$$\mathcal{K}\Phi(x_k) = \lambda\Phi(x_k) = \Phi(x_{k+1}) \tag{8}$$

此时，本征函数就会成为观测空间的一组基底，由此，数学上，观测函数应当表示为这组基底的线性组合，如下：

$$g(x) = \sum_{i=1}^{n} a_i\Phi_i(x) \tag{9}$$

此时，观测空间可被称为状态字典空间，同时，空间的基底可被称为字典函数

当前，从数据中挖掘信息并获得 Koopman 本征函数是现代动力学系统研究的主流方法，被称为数据驱动（Data-Driven）的 Koopman 算子。由此，我们可以通过数据驱动的方式得到 Koopman 算子的本征函数，以得到非线性系统在高维空间中的全局线性表示。

此外，Koopman 算子已经被广泛运用于受控系统。在受控确定性离散时间系统中，我们有：

$$x' = F(x,u) \tag{10}$$

在受控连续时间系统中：

$$\frac{\mathrm{d}}{\mathrm{d}t}x = f(x(t),u(t)) \tag{11}$$

如果考虑到 $u_{k+1} = u_k$ 的控制输入，我们可以将等式 (7) 修改为：

$$g(x_{k+1},u_{k+1}) = \mathcal{K}g(x_k,u_k) \tag{12}$$

其中 $x \in \mathbb{R}^n, u \in \mathbb{R}^m$，且 $g : \mathbb{R}^{n+m} \mapsto \mathbb{R}^d$。

2.1.3 数据驱动的 Koopman 算子

数据驱动的 Koopman 算子是一种从系统数据中学习系统动力学行为的方法。它的主要思想是利用系统的观测数据来构建 Koopman 算子的近似，而无需事先了解系统的动力学方程。具体来说，给定系统的观测数据序列 $x_0, x_1, \ldots, x_N$，其中 x_i 是系统在时间 t_i 的状态观测值，数据驱动的 Koopman 算子可以通过以下步骤获得：

动态模态分解

动态模态分解 (Dynamic Mode Decomposition, DMD) 是一种基于数据的模型约简技术，用于从系统观测数据中提取动力学模式。给定系统的观测数据序列 $x_0, x_1, \ldots, x_N$，DMD 可以分解系统的状态演化矩阵 A，并提取系统的动态模态。具体步骤如下：构建数据矩阵 X 和下一时刻数据矩阵 X'。对数据矩阵 X 和 X' 进行奇异值分解 (SVD)，得到矩阵 U, Σ, V。估计系统状态演化矩阵 A。计算系统的动态模态，即 A 的特征向量。DMD 可以用于系统预测、模型约简和特征提取等应用，在流体力学、结构动力学和气象学等领域得到广泛应用。

扩展动态模态分解

扩展动态模态分解 (Extended Dynamic Mode Decomposition, EDMD) 是 DMD 的一种扩展形式，用于处理非线性系统的观测数据。与传统的 DMD 相比，EDMD 引入了非线性特征映射，以便处理非线性系统的演化。具体步骤如下：构建非线性特征映射，将观测数据映射到高维特征空间。在高维特征空间中构建数据矩阵 X 和下一时刻数据矩阵 X'。对数据矩阵 X 和 X' 进行奇异值分解 (SVD)，得到矩阵 U, Σ, V。估计系统状态演化矩阵 A。计算系统的动态模态，即 A 的特征向量。EDMD 通过引入非线性特征映射，可以更好地处理非线性系统的观测数据，并提取系统的动态模态。

2.2 强化学习

强化学习（Reinforcement Learning）是机器学习（Machine Learning）和控制理论（Control Theory）的交叉领域，在强化学习中，智能体学习如何与复杂环境进行交互，并以获得更高的奖励为目标。最近，深度强化学习（Deep Reinforcement Learning）被证明能够在若干项具有挑战性的任务中，实现人类水平或超人类的表现，包括玩电子游戏 [cite] 和策略游戏 [cite]。深度强化学习也越来越多地用于科学和工程应用，包括药物发现 [cite]、机器人操作 [cite]，自动驾驶 [cite] 和无人机赛车 [cite]、流体流量控制 [cite] 和融合控制 [cite]。

2.2.1 马尔可夫决策过程与贝尔曼方程

马尔可夫决策过程是具有马尔可夫性质的随机过程。

随机过程是指研究对象是随时间演变的随机现象，在随机过程中，随机现象在某一时刻 t 的取值是一个向量随机变量，可以用 S_t 表示，所有可能的状态组成状态空间 $\mathcal{S}$。我们将已知所有历史状态 $(S_1,\ldots,S_t)$ 时，某一时刻 t 的状态 S_t 发生的概率用 $P(S_{t+1}|S_1,\ldots,S_t)$ 表示。而马尔可夫性质则表示，已知当前时刻状态 S_t 时，下一时刻状态 S_{t+1} 仅与 S_t 有关，用 $P(S_{t+1}|S_t)$ 表示。而从前一状态经过随机进入下一状态的过程被称为状态转移。

在下文中，我们假设存在一个无限时域的马尔可夫决策过程。我们假设代理跟随随机策略 $\pi(u|x)$，表示在已知状态 x 的情况下，采取某个特定动作 u 的可能性。由此，在离散时间系统中，状态价值函数定义为：

$$V^{\pi}(x)=\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^{k}r(x_k,u_k)|\pi,x_0=x\right] \tag{13}$$

其中，$r(x_k,u_k)$ 表示智能体得到的奖励；$\gamma\in[0,1]$ 表示折扣率，用于避免在无限时域时无限大的奖励。

在这里我们强调强化学习与控制领域的结合，所以本文中考虑使用线性二次最优控制问题。线性二次最优控制是十分经典的控制领域问题，其中线性代表研究的系统动态可以用一组线性微分方程表示，而其成本为二次泛函。形式上，我们考虑有限时间长度，离散时间的 LQR，假设离散时间线性系统：

$$x_{k+1}=Ax_k+Bu_k$$

其性能指标为：

$$c(x_k,u_k)=x_k^{T}Qx_k+u_k^{T}Ru_k \tag{14}$$

将系统线性二次性能指标的相反数作为智能体探索时的奖励:

$$r(x_k,u_k)=-c(x_k,u_k) \tag{15}$$

并改写贝尔曼方程可以得到：

$$V^{\pi}(x)=\mathbb{E}\left[\sum_{k=0}^{\infty}-\gamma^{k}c(x_k,u_k)|\pi,x_0=x\right] \tag{16}$$

2.2.2 演员-评委强化学习算法

演员-评论家（Actor-Critic）强化学习算法是一种结合价值学习和策略学习的强化学习算法，下面分三步介绍 Actor-Critic 强化学习算法。

首先介绍策略学习，在策略学习中，评价一个策略的方法是评估策略在环境中获得的回报的期望，此时目标函数为：

$$J(\theta) = \mathbb{E}_S [V_\pi(S)] \tag{17}$$

其中，V_π 代表状态价值函数，可以看到在这个目标函数中，求得了状态价值函数有关状态的期望，所以只涉及到策略 π 的信息。同时易知，这个目标函数该期望越大，表明该策略越好。

注意，在本文中，对于强化学习的研究只涉及深度强化学习，所以在这里使用深度神经网络的方式对策略网络进行测试。所以，问题转化为如下的一个最大化问题：

$$\max_\theta J(\theta) \tag{18}$$

很明显，我们只需要使用梯度上升的方式迭代策略网络。参数更新如下：

$$\theta_{new} = \theta_{old} + \beta \cdot \nabla_\theta J(\theta_{now}) \tag{19}$$

其中，β 代表学习率，是需要调整的超参数，梯度的具体表达式如下：

$$\nabla_\theta J(\theta_{now}) \triangleq \frac{\partial J(\theta)}{\partial \theta} \tag{20}$$

以上算法被称为策略梯度

在策略学习中，我们想要得到的是当前策略能得到的价值函数，表示如下：

$$V^\pi(s_t) = \mathbb{E}\left[\sum_{k=t}^{N} \gamma^{k-t} \cdot r_k\right] \tag{21}$$

在深度强化学习中，我们通过深度神经网络对价值进行近似，算法上，应该使得上述价值网络逼近真实价值函数。一般使用时间差分（Temporal Differencial）算法来更新价值函数，根据贝尔曼方程：

$$V^\pi(s_t) = \mathbb{E}[r_t + \gamma V^\pi(s_{t+1})] \tag{22}$$

等式左边为价值函数处于时间步 t 时对于价值的估计，等式右边是对于左边的一个无偏估计，其中包含的 r_t 项使得等式右边比左边更加精确，所以应当使得价值网络逼近等式右边的值。参数更新如下：

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) Q^{now}(s_t, a_t) + \alpha[r_t + \gamma Q^{now}(s_{t+1}, a_{t+1})] \tag{23}$$

最后，通过交替执行策略优化和价值优化，可以使得两者的性能逐渐同步提升。策略网络可以获得更好的状态价值，价值网络可以实现更好的价值估计。最后取用策略网络作为最优控制的策略函数就可以完成 Actor-Critic 算法在最优控制问题上的应用。

2.3 神经网络

本文中将使用简单的神经网络对于函数进行近似，在本节中将对于神经网络进行一些简单的介绍。

2.3.1 简单架构

神经网络架构的灵感来自与人脑中的神经元，它们构成了一个复杂的互相连接的神经元网络，通过互相发送电信号来交换和处理信息。同样的，人工神经网络由人工神经元构成，以相似的结构结合在一起，共同合作以解决问题。

实际上，人工神经元是以层为单位结合在一起的，基本神经网络主要分为三层：

输入层

来自神经网络外部的信息会通过输入层进入人工神经网络。输入节点对数据进行处理、分析或者分类，然后继续传递到下一层。

隐藏层

隐藏层从输入层或者其他隐藏层获取其输入。人工神经网络可以具有大量的隐藏层，每个隐藏层都会对来自上一层的输出进行分析和进一步处理，然后将其继续传递到下一层。

输出层

输出层提供人工神经网络对所有数据进行处理的最终结果。它可以包含单个或多个节点。例如，如果我们要解决一个二元（是/否）分类问题，则输出层包含一个输出节点，它将提供 1 或 0 的结果。但是，如果我们要解决一个多类分类问题，则输出层可能会由一个以上输出节点组成。

下面介绍全连接层：令输入向量 $x \in \mathbb{R}^d$，神经网络的一个层把 x 映射为 $x' \in \mathbb{R}^{d'}$。全连接层定义如下：

$$x' = \sigma(z), \qquad z = Wx + b \tag{24}$$

其中，权重矩阵 $W \in \mathbb{R}^{d' \times d}$ 和偏置向量 $b \in \mathbb{R}^{d'}$ 是该层的参数，需要从数据中学习；$\sigma(\cdot)$ 是激活函数，比如 softmax 函数、sigmoid 函数，ReLU（Rectified Linear Unit）函数。最常用的激活函数是 ReLU。定义如下：

$$\mathrm{ReLU}(z) = [(z_1)_+, (z_2)_+, \cdots, (z_{d'})_+,]^T \tag{25}$$

上面的 $[z_i]_+ = \max(z_i, 0)$。我们称上面的结构为全连接层（Fully Connected Layer）。如果把全连接层当作基本部件，然后像搭积木一样搭建一个全连接神经网络，也叫多层感知机（Multi-Layer Perceptron, MLP）

2.3.2 反向传播

线性模型和神经网络的训练都可以描述成一个优化问题，设全连接层的权重参数为 $\omega^{(1)},\cdots,\omega^{(l)}$ 我们希望求解下面这样的一个优化问题：

$$\min_{\omega^{(1)},\cdots,\omega^{(l)}} L(\omega^{(1)},\cdots,\omega^{(l)}) \tag{26}$$

其中 L 表示损失函数。对于这样一个无约束的最小化问题，最常使用的方法是梯度下降（Gradent Descent, GD）和随机梯度下降（Stochastic Gradient Descent, SGD），本节主要介绍使用 SGD 求解最小化问题。

假设目标函数可以写成 n 连加的形式：

$$L(\omega^{(1)},\cdots,\omega^{(l)}) = \frac{1}{n}\sum_{j=1}^{n} F_j(\omega^{(1)},\cdots,\omega^{(l)}) \tag{27}$$

函数 F_j 隐含第 j 个训练样本 (x_j,y_j)。每次从集合 $\{1,2,\cdots,n\}$ 中随机抽取一个数，记作 j。当前的参数只为 $\omega_{now}^{(1)},\cdots,\omega_{now}^{(j)}$，计算此处的梯度，SGD 算法的迭代过程为

$$\omega_{new}^{(i)} \leftarrow \omega_{now}^{(i)} - \alpha \cdot \underbrace{\nabla\omega^{(i)} F_j\left(\omega_{now}^{(1)},\cdots,\omega_{now}^{(l)}\right)}_{\text{随机梯度}}, \quad \forall i = 1,\cdots,l. \tag{28}$$

下面我们介绍反向传播（Backpropagation，BP），以全连接神经网络为例，简单介绍反向传播的原理。全连接神经网络（忽略偏移量 b）定义如下：

$$\begin{aligned} x^{(1)} &= \sigma_1(W^{(1)}x^{(0)}),\\ x^{(2)} &= \sigma_2(W^{(2)}x^{(1)}),\\ &\vdots\\ x^{(l)} &= \sigma_l(W^{(l)}x^{(l-1)}), \end{aligned}$$

神经网络的输出 $x^{(l)}$ 是神经网络做出的预测。设向量 y 为真实标签，函数 H 为交叉熵，实数 z 为损失函数：

$$z = H(y,x^{(l)}) \tag{29}$$

要做梯度下降更新参数 $W^{(1)},\cdots,W^{(l)}$，我们需要计算损失函数 z 关于每一个变量的梯度：

$$\frac{\partial z}{\partial W^{(1)}}, \frac{\partial z}{\partial W^{(2)}}, \cdots, \frac{\partial z}{\partial W^{(l)}} \tag{30}$$

现在可以用链式法则做反向传播，计算损失函数 z 关于神经网络参数的梯度。具体地，首先求出梯度 $\frac{\partial z}{\partial x^{(l)}}$。然后做循环，从 $i = l,\cdots,1$，依次执行如下操作：根据链式法则可得损失函数 z 关于参数 $W^{(i)}$ 的梯度：

$$\frac{\partial z}{\partial W^{(i)}} = \frac{\partial x^{(i)}}{\partial W^{(i)}} \cdot \frac{\partial z}{\partial x^{(i)}} \tag{31}$$

此梯度用于更新参数 $W^{(i)}$。根据链式法则可得损失函数 z 关于参数 $x^{(i-1)}$ 的梯度:

$$\frac{\partial z}{\partial x^{(i-1)}} = \frac{\partial x^{(i)}}{\partial x^{(i-1)}} \cdot \frac{\partial z}{\partial x^{(i)}} \tag{32}$$

这个梯度被传播到下面一层（即第 $i-1$ 层）, 继续循环。

综上所述，只要知道损失函数 z 关于 $x^{(i)}$ 的梯度，就能求出 z 关于 $W^{(i)}$ 和 $x^{(i-1)}$ 的梯度。

第三章 Koopman 算子辅助的最大熵强化学习

贝尔曼方程及其连续形式汉密尔顿-雅可比-贝尔曼（Hamilton-Jacobi-Bellman）方程在强化学习和控制理论中普遍存在。然而，这些方程对于具有高维状态和非线性的系统来说，很快就变得难以处理。本章将讨论数据驱动的 Koopman 算子与马尔可夫决策过程之间的联系，并介绍借由此开发的两种新的强化学习算法来解决上述的问题。

技术上，我们可以利用 Koopman 算子技术将非线性系统提升到新的坐标中，其中动力学变得近似线性。特别是，Koopman 算子能够通过提升坐标中的线性动力学来捕获给定系统的价值函数的时间演化的期望。

3.1 动作加权的 Koopman 算子

我们现在关注如何通过 Koopman 算子在时间上推进观测基底函数，已知当前状态和动作，也就是说，找到下面的一个映射：

$$(x,u) \mapsto \phi(\dot{x}) \tag{33}$$

我们要求观测函数近似地覆盖每个控制变量 u 的价值函数的有限维的 Koopman 不变子空间，因此存在矩阵 $K^u \in \mathbb{R}^{d_x \times d_x}$，使得 $K^u \phi(x) = \phi(\dot{x})$。

给定轨迹数据和状态字典空间 ϕ，为每个动作 $u \in \mathcal{U}$ 构建如上的 Koopman 矩阵 K^u。为此，采取类似于 SINDYc[cite] 中描述的方法，其中状态和动作的字典空间用于预测下一个状态，即 $\Theta(x,u) \mapsto \dot{x}$ 而这里的方法不同的地方有两点。首先，不是试图预测下一个状态 x，而是试图预测下一个字典函数值 $\phi(x)$。其次，为了让状态字典空间覆盖每个 u 的 Koopman 不变子空间，状态-动作字典在状态和动作上是可分离的。

这里，将状态-动作字典空间建模为乘法可分离的，就像 $\psi(x) \otimes \phi(x)$ 然后进一步假设存在线性映射 $\psi(x) \otimes \phi(x) \mapsto \phi(\dot{x})$。下面描述如何为任何的 u 构建矩阵 K^u。

令 $\phi : \chi \mapsto \mathbb{R}^d_x$ 为状态的特征映射（ϕ 的每一个分量都是一个可观测函数），并且令 $\psi : \mathcal{U} \mapsto \mathbb{R}^d_u$ 为控制动作的特征映射。算法要求对于所有 $u \in \mathcal{U}$ 得到 Koopman 算子 $\mathcal{K}^u$ 的有限维逼近。令 $\mathcal{T}_K \in \mathbb{R}^{d_x \times d_x \times d_u}$ 为如图 [ref] 所示的三维向量。对于任何 u，如下定义：

$$K^u[i,j] = \sum_{z=1}^{d} \mathcal{T}_K(i,j,k)\phi(u)[z] \tag{34}$$

形式上，K^u 是沿着张量 $\mathcal{T}$ 第三维的向量积，同时，K^u 作为对于 Koopman 算子 $\mathcal{K}^u$ 的有限维近似。因此，我们可以通过学习 $\mathcal{T}_K$，来最小化字典函数 ϕ 的误差：

$$\min_{\mathcal{T}_K} \sum_{i=1}^{N} ||K^{u_i}\phi(x_i) - \phi(x_i')||^2 \tag{35}$$

我们可以重写上述目标，使其成为一个规则的多变量线性回归问题，将 $\mathcal{T}_K$ 重新排列为一个

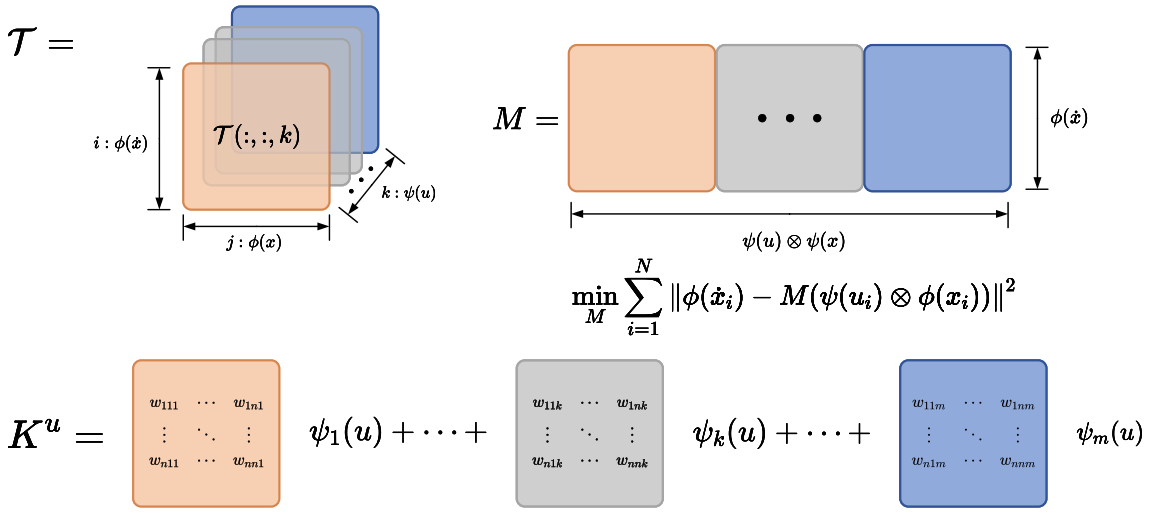

图 3.1　Koopman 张量估计

大小为 $\mathbb{R}^{d_x \times d_x d_u}$ 的二维矩阵。令 $M \in \mathbb{R}^{d_x \times d_x d_u}$，其中 $M[i,:] \in \mathbb{R}^{d_x \cdot d_u}$ 是堆叠二维矩阵 $\mathcal{T}_K[i,:,:]$ 所得到的向量，如图 [3.1] 所示。令 $\psi(u) \otimes \phi(x) \in \mathbb{R}^{d_x d_u}$ 为克罗内克积（Kronecker Product），所以有：

$$K^u \phi(x) = M(\psi(u) \otimes \phi(x)) \tag{36}$$

从而，优化问题变成了常规的线性规划问题：

$$\min_M \sum_{i=1}^{N} ||M(\phi(u) \otimes \phi(x_i)) - \phi(\dot{x}_i)||^2 \tag{37}$$

算法见算法 1 。

只要我们计算出了 M，我们就可以通过将 M 转换回为三维张量 $\mathcal{T}_K$ 的方式为任意系统执行的 $u \in \mathcal{U}$ 得到标准的 Koopman 算子，同时也得到了上述对 $\mathcal{K}^u$ 的有限维近似 $K^u \in \mathbb{R}^{d_x \times d_x}$

算法 1 Koopman 张量估计

输入： 状态特征映射 $\phi : \chi \mapsto \mathbb{R}^{d_x}$，控制特征映射 $\psi : \mathcal{U} \mapsto \mathbb{R}^{d_u}$ 以及一个样本集合：$\{(x_i, u_i)\}_{i=0}^{N}$

1: 从等式 (37) 中解出 $\hat{M}$

2: 以 Fortran 风格将 M 转换成 $\mathcal{T}_K$

3.2　最大熵强化学习最优控制问题

考虑强化学习在最优控制领域的应用，在实践中最大的问题有以下两点：首先，非常高的样本复杂度：使得智能体在探索中遇到困难；其次，缓慢、不稳定的收敛过程：探索

和利用的权衡是策略优化的重点，特别是在最优控制中，不合适的样本利用和探索会使得策略过早的收敛到不良的局部最优值。本文中使用最大熵强化学习算法，通过在训练目标中合理加入熵正则化项。经过训练，策略可以最大化预期回报和熵。

3.2.1 熵正则化

粗略的说，熵是一个表示随机变量随机程度的量。具体的说，如果一个随机变量的概率分布很集中，那么这个随机变量的熵就低；如果一个随机变量的概率分布比较分散，那么这个随机变量的熵就高。

令 x 是具有概率质量密度函数的随机变量 P。H 的熵是根据 x 的分布得出的：

$$H(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)]$$

在熵正则强化学习中，代理在每个时间步获得与该时间步的策略熵成比例的奖金。由此，目标策略变为：

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t))\right)\right] \tag{38}$$

其中 $\alpha > 0$ 是权衡参数。我们现在可以在如上假定中给出略有不同的价值函数。V^π 更改为包含每个时间步的熵奖励：

$$V^\pi = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t)) \mid s_0 = s\right)\right] \tag{39}$$

Q^π 更改为包括除了第一个时间步以外的每个时间步的熵奖励：

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H(\pi(\cdot|s_t)) \mid s_0 = s, a_0 = a\right] \tag{40}$$

通过这些定义，V^π 和 Q^π 通过以下方程联系起来：

$$\begin{aligned} V^\pi(s) &= \mathop{\mathbb{E}}_{\dot{s}\sim P,\dot{a}\sim\pi}[r(s,a,\dot{s}) + \gamma(Q^\pi(\dot{s},\dot{a}) + \alpha H(\pi(\cdot|\dot{s})))] \\ &= \mathop{\mathbb{E}}_{\dot{s}\sim P}[r(s,a,\dot{s}) + \gamma V^\pi(\dot{s})] \end{aligned} \tag{41}$$

标准的强化学习最大化奖励之和的期望，但是在加入熵正则方法的强化学习中，将会考虑一个更加一般的最大熵目标 [cite]：

$$J(\pi) = \sum_{t=0}^{T}\mathbb{E}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{42}$$

权重参数 α 决定了熵项对于奖励的相对重要性，从而控制了最优策略迭代中的的随机性。

3.2.2 Soft Actor-Critic 强化学习最优控制算法

下面，介绍如何通过策略迭代公式设计 SAC 算法，我们将从推导熵策略迭代（Soft Policy Iteration）开始，这是一种在应用了熵正则的方法下学习最优策略的通用算法，在最大熵框架中交替进行策略评估和策略改进。

在策略评估步骤中，我们希望根据等式 (42) 中的最大熵目标计算策略 π 的值。对于固定策略，可以从任何函数 $Q:\mathcal{S}\times\mathcal{A}\mapsto\mathbb{R}$ 开始，并且使用下面的修改过的贝尔曼方程迭代计算熵动作价值：

$$Q(s_t,a_t)=r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim p}\left[V(s_{t+1})\right] \tag{43}$$

在策略改进步骤中，策略将逼近于新的熵动作价值函数的指数。如下的这种特定的设定可以保证策略的改进 [cite]：

$$\pi_{new}=\arg\min_{\pi'\in\Pi}\mathrm{D}_{KL}\left(\pi'(\cdot|s_t)\middle\|\frac{\exp(Q^{\pi_{old}}(s_t,\cdot)}{Z^{\pi_{old}}(s_t)}\right) \tag{44}$$

尽管该算法将可证明地找到最优解，但我们只能在表格情况（有限个状态和动作）下以其精确形式执行它。因此，我们接下来将对连续域进行估计，其中我们需要依靠函数逼近器来表示动作价值函数，这种近似的做法产生了一种新的实用算法，称为 Soft Actor-Critic 算法。

如上所述，大型连续域要求我们推导出熵策略迭代的实际逼近。为此，我们将对动作价值函数和策略使用函数逼近器，并且交替使用随机梯度下降（Stochastic Gradient Descent）来优化这两个网络。我们将考虑参数化状态价值函数 $V_\psi(s_t)$、熵动作价值函数 $Q_\theta(s_t,a_t)$ 和策略 $\pi_\phi(a_t|s_t)$。这些网络的参数是 ψ,θ 和 ϕ。下面介绍这些参数向量的更新规则。

设定如下熵状态函数训练目标，以最小化平方残差：

$$J_V(\psi)=\mathop{\mathbb{E}}_{s_t\sim D}\left[\frac{1}{2}(V_\psi(s_t)-\mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t)-\log\pi_\phi(a_t,|s_t)\right])^2\right] \tag{45}$$

其中，D 是先前采样状态和动作的概率分布，或者是经验回放。等式 (45) 的梯度可以用下面的无偏估计表示：

$$\hat{\nabla}J_V(\psi)=\nabla_\phi V_\phi(s_t)(V_\phi(s_t)-Q_\theta(s_t,a_t)+\log\pi_\phi(a_t|s_t)) \tag{46}$$

其中，动作是根据当前策略采样的，而不是从经验回放中抽取。可以如下训练熵动作价值函数参数来最小化熵贝尔曼残差：

$$J_Q(\theta)=\mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_\theta(s_t,a_t)-\hat{Q}(s_t,a_t))^2\right] \tag{47}$$

其中

$$\hat{Q}(s_t,a_t)=r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim p}\left[V_{\overline{\psi}}(s_{t+1})\right] \tag{48}$$

同样可以用随机梯度下降优化：

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_{\overline{\phi}}(s_{t+1})) \tag{49}$$

更新利用了目标值网络 $V_{\overline{\psi}}$，其中 $\overline{\psi}$ 在更新参数时加权更新，这种方式已被证明可以稳定训练 [cite]。最后，可以通过直接最小化等式 (44) 中的 KL 散度（Kullback-Leibler Divergence）来学习策略参数：

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D}\left[D_{KL}\left(\pi_\phi(\cdot|s_t)\middle\|\frac{\exp(Q_\theta(s_t,\cdot)}{Z_\theta(s_t)}\right)\right] \tag{50}$$

首先使用神经网络变换来参数化策略：

$$\mathbf{a}_t = f_\phi(\epsilon_t; \mathbf{s}_t) \tag{51}$$

其中 t 是输入噪声向量，从一些固定分布中采样，例如球面高斯。我们现在可以将等式 (50) 中的目标重写为：

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D, \epsilon_t \sim N}\left[\log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))\right] \tag{52}$$

我们可以近似等式 (52) 的梯度：

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t} \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a_t}} Q(\mathbf{s}_t, \mathbf{a}_t))\nabla_\phi f_\phi(\epsilon_t; \mathbf{s}_t) \tag{53}$$

其中，a_t 由 $f_\phi(\epsilon_t; \mathbf{s}_t)$ 评估。

完整的算法见算法 **2** 。

算法 2 Soft Actor-Critic 强化学习算法

初始化参数向量 $\psi, \overline{\psi}, \theta, \phi$
for 每一次迭代 **do**
  for 每一个时间步 **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $D \leftarrow D \cup (\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})$
  end for
  for 每一次梯度 **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\phi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\phi)$ for $i \in 1, 2$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\overline{\psi} \leftarrow \tau\psi + (1-\tau)\overline{\psi}$
  end for
end for

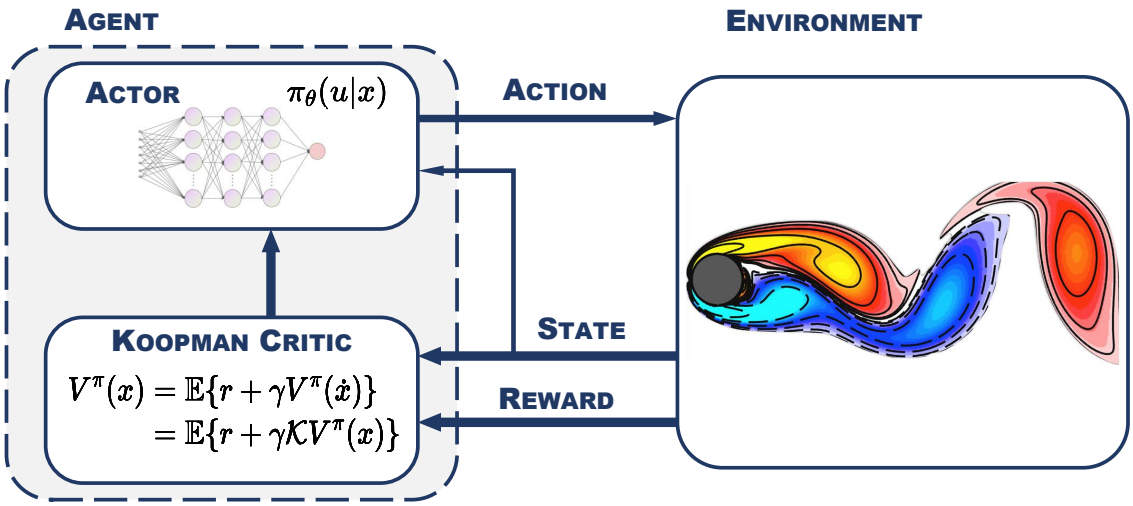

图 3.2 Soft Actor Koopman-Critic

下面，介绍如何通过修改 Soft Actor-Critic 框架 [cite]，以合并来自 Koopman 算子的信息来限制探索空间的。

这里使用与 Soft Actor-Critic 论文 [53] 相同的损失函数和类似的符号，我们首先指定熵价值函数损失：

$$J_V(\psi) = \mathop{\mathbb{E}}_{s_t \sim D}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}\left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t, |s_t)\right])^2\right] \tag{54}$$

在这里，介绍有关 Koopman 熵价值函数的新定义：

$$V_\psi(x) = \psi^T \phi(x) \tag{55}$$

其中，ω 是字典函数的系数向量，表示价值函数在是字典函数也就是观测函数的线性组合。也就是说，价值函数的评估是集中在 Koopman 升维后的观测空间中进行的。同时，我们知道，合理的观测空间应该是由一组完备的观测函数所构建的，所以在价值评估的准确性上是有理论支撑的。

下面，介绍如何改动动作价值函数的目标函数：

$$J_Q(\theta) = \mathbb{E}_{(x,u)\sim D}\left[\frac{1}{2}\left(Q_\theta(x,u) - \hat{Q}(x,u)\right)^2\right] \tag{56}$$

其中，训练目标融合了 Koopman 算子，被定义为：

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}\left[V_{\overline{\omega}}(s_{t+1})\right] \tag{57}$$

在这里，考虑价值预测向量 $\overline{\omega}$ 和 Koopman 算子，将上式记为：

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \overline{\omega}^T K^u \phi(s_t) \tag{58}$$

最后，策略目标函数并没有更改：

$$J_\pi(\nu) = \mathbb{E}_{s\sim D}\left[D_{KL}\left(\pi_\nu(\cdot|s)\middle\|\frac{\exp(Q_\theta(s,\cdot))}{Z_\theta(s)}\right)\right] \tag{59}$$

通过上述修改，实际算法在表述上与算法 2一致。而不同之处已经在上述公式中解释清楚了。

Koopman 辅助的最大熵强化学习以 Soft Actor Koopman-Critic 为例，创建了 Koopman Critic，其本质是流行的 Soft Actor-Critic 算法的 Koopman 变体。

Koopman Critic 接收状态和奖励作为原始的非线性动态，然后将这些动态提升到向量空间，在那个空间里，系统状态可以通过 Koopman 算子线性地推进。然后将此批评反馈给 Actor，Actor 据此计算要在环境中执行的动作。整体的框架如图 3.2所示。

3.3 本章小结

本章主要介绍一种用于非线性系统最优控制系统的，基于 Koopman 算子辅助强化学习训练的算法。该算法的研究对象主要是第二章中提及的受控非线性系统，训练算法为最大熵强化学习。

同时，通过 Koopman 算子理论，将最大熵强化学习中的价值函数，在贝尔曼方程中进行代换的方式，得到贝尔曼方程中价值函数有关时间序列的期望，这样做的好处是提高算法对于非线性系统的适应性，同时保留了 Koopman 算子对于非线性系统动态的洞察，保留了输入输出的可解释性。

第四章 深度 Koopman 算子网络非线性系统最大熵强化学习

在本章中，首先介绍一个端到端的神经网络学习框架，用于替代上文介绍的其他数据驱动的 Koopman 算子，由于该神经网络架构额外的对于高维度系统动力学信息更好的把握，可以期望其得到更精准的预测。同时，将其与 Koopman 辅助的强化学习结合，得益于更加精确的价值函数估计，可以使得最大熵强化学习效果更好，训练更快。此外，本章中还提出一种 Koopman 算子和强化学习交替训练的训练架构，更好的发挥神经网络的优势使得训练流程更加简化，对于样本的利用更加有效。

4.1 深度 Koopman 算子网络

我们首先将嵌入函数分为两部分：在不失一般性的情况下，我们假设第一项仅与系统状态相关，$g(x,u)=[g_x(x,u);g_u(x,u))]$ 所以我们得到 $g_x(x,u)=g_x(x)$。基于 Koopman 算子的定义，将 Koopman 算子 $\mathcal{K}$ 拆分成两个矩阵，则可以得出：

$$g_x\left(x_{k+1}\right)=K_{xx}g_x\left(x_k\right)+K_{xu}g_u\left(x_k,u_k\right) \tag{60}$$

对于嵌入函数的第一部分，我们使用神经网络 θ 来参数化 $g_x(x,k)$，将原始状态和网络编码连接起来在一起:

$$z_k=g_x\left(x_k\right)=\begin{bmatrix} x_k \\ g_\theta\left(x_k\right) \end{bmatrix} \tag{61}$$

其中 z_k 是升维系统（这里也叫嵌入空间）状态，$g_\theta:\mathbb{R}^n\mapsto$ 是参数神经网络。这种设计的优点是可以通过以下方式轻松恢复原始状态：

$$x_k=Cz_k \tag{62}$$

其中，矩阵 $C\in\mathbb{R}^{n\times(n+d)}$ 的一个样板如下：

$$C=[I_n \quad 0] \tag{63}$$

这样，我们就可以保留成本函数的形式进行进一步的控制。

对于嵌入函数的第二部分，在深度 Koopman 算子理论中，有三个版本来表示 $g_u(x_k,u_k)$。

第一个版本被称为深度 KoopmanU（Deep KoopmanU with Control, DKUC）算法。其将函数简化为 $g_u(x_k,u_k)=u_k$，然后演化方程退化为 $g_x(x_{k+1})=K_{xx}g_x(x_k)+K_{xu}u_k$。

第二个版本被称为深度 Koopman 仿射（Deep Koopman Affine with Control, DKAC）算法，它将进化函数视为控制仿射形式，使得 $g_u(x_k,u_k)=g_u(x_k)u$，并且将等式 (7) 修改为：

$$z_{k+1}=K_{xx}z_k+K_{xu}g_u(x_k)u \tag{64}$$

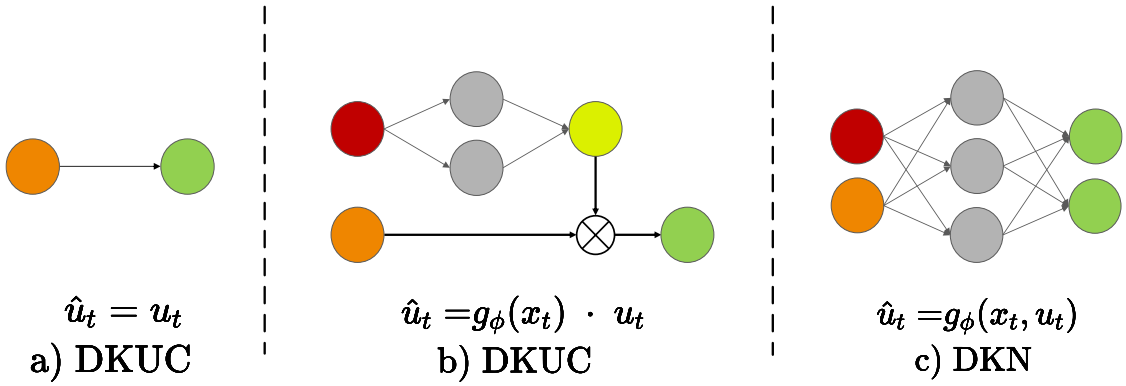

图 4.1 不同形式的控制网络示意图

对于最终版本，使用控制网络 ϕ 将函数 $g_u(x_k,u_k)$ 直接近似为 $g_u(x_k,u_k)=g_\phi(x_k,u_k)$，并且演化函数保持与等式 (7) 相同的形式。它被称为深度 Koopman 非线性（Deep Koopman Nonlinear, DKN）算法。

给定当前状态 x_t，我们可以很容易地通过前馈网络预测未来的 K 步状态。除了嵌入函数外，我们还通过一层线性网络将 Koopman 算子矩阵 K_{xx} 和 K_{xu} 参数化，即令 $K_{xx},K_{xu}=B$。然后 K 步预测如下：

$$\begin{aligned} z_{t+k+1} &= Az_{t+k} + Bg_\phi\left(x_{t+k},u_{t+k}\right), k=0,\ldots,K-1 \\ z_t &= g_x\left(x_t\right) = \begin{bmatrix} x_t \\ g_\theta\left(x_t\right) \end{bmatrix} \\ x_{t+k} &= Cz_{t+k} \end{aligned} \tag{65}$$

其中，对于 DKUC 算法，$g_\phi(x,u)=u$，对于 DKAC 算法，$g_\phi(x,u)=g_\phi(x)u$，对于 DKN 算法，$g_\phi(x,u)=g_\phi(x,u)$。网络示意如图 (4.1)

Koopman 的标准算法应首先选择嵌入函数，然后使用线性回归学习 Koopman 算子矩阵，这种方法仅用于单步预测。在这项工作中，我们端到端地学习嵌入函数和 Koopman 算子，并设计了用于长期预测的 K 步预测损失函数。给定数据集 $[X_i\in\mathbb{R}^{N\times n},U_i\in\mathbb{R}^{N\times m},i=0,1,2,\cdots,K]$ 我们可以计算实际嵌入状态 $Z_i=g_x(X)$，并通过方程 (65) 从初始状态 X_0 预测嵌入空间中的 K 阶状态 $[\hat{Z}_i,i=1,2,\cdots,K]$。损失函数计算如下：

$$L(\theta,\phi)=\sum_{i=1}^{K}\gamma^{i-1}\,\mathrm{MSE}\left(Z_i,\hat{Z}_i\right) \tag{66}$$

其中 是权重衰减超参数，MSE 是均方损失函数 (Mean Square Error)。K 步损失不是只考虑下面的一步预测误差，而是关注 K 步预测误差的加权和，有利于在长时间范围内进行预测。

4.2 基于深度 Koopman 算子网络的非线性系统最大熵强化学习

上一节介绍了深度 Koopman 算子是如何通过网络架构和而在这一节中，我们介绍如何将深度 Koopman 算子和最大熵强化学习结合，以求使得我们的最优控制算法得到更好的收敛速度和训练效果。

在第三章中，介绍了动作加权的 Koopman 算子，其目的是为了更好的获取系统动作的信息，不同的动作量信息，在做单步预测时，应当在 Koopman 算子中体现出来，以求达到更好的预测能力。

在实际的控制中，动作变量可能与系统的多个变量有着联系, 甚至其中的数学关系不能够用线性关系来描述，很明显，简单的加权计算并不能够表述这样的关系。这就会导致动作的轨迹在升维空间或称之为嵌入空间中会发生偏移和失真。也就简介导致了 Koopman 价值函数在这种情况下会发生误差，导致学习过程受影响。

下面介绍深度 Koopman Critic 网络来解决上面的问题：

我们假设有嵌入网络 $g_\theta : d_x \mapsto d_z$，控制辅助网络 g_ϕ，Koopman 算子矩阵 A, B 和恢复矩阵 C。下面介绍网络架构和前馈过程：

有系统状态和输入：(x_t, u_t)，嵌入网络将系统状态提升到嵌入空间：

$$z_t = [\ x_t^T \quad g_\theta(x_t)^T\ ]^T \tag{67}$$

同样的，辅助控制网络将系统控制提升到嵌入空间：

$$\hat{u_t} = g_\phi(x_t, u_t) \tag{68}$$

这里，使用了深度 Koopman 非线性形式的辅助网络，使用了当前时刻系统状态 x_t 和系统控制 u_t 作为输入信息，在使用了神经网络参数化的策略函数的情况下，保证了可以准确捕捉动作信息与状态信息之间的关系。

下面介绍两个 Koopman 算子分量矩阵 A, B:

$$z_{t+1} = Az_t + B\hat{u}_t \tag{69}$$

可以看出，在这里我们使用了两个 Koopman 矩阵，并且取两个矩阵的和作为新的升维状态，同时考虑了状态信息和动作信息。具体网络架构如图 (A.1) 所示。

在此基础上，我们考虑如等式中的 (57) Koopman-Critic 函数，将 Koopman 算子和嵌入函数结合的函数 $K^u\phi(\cdot)$ 通过上述的神经网络结合，将等式 (57) 改写为：

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma\overline{\omega}^T(Az_t + B\hat{u}_t) \tag{70}$$

在上面的修改下，我们依然可以遵从算法 2 对参数网络进行迭代。

参考文献

[1] Tenne R, Margulis L, Genut M e, et al. Polyhedral and cylindrical structures of tungsten disulphide[J]. Nature, 1992, 360(6403):444–446.

附　　录

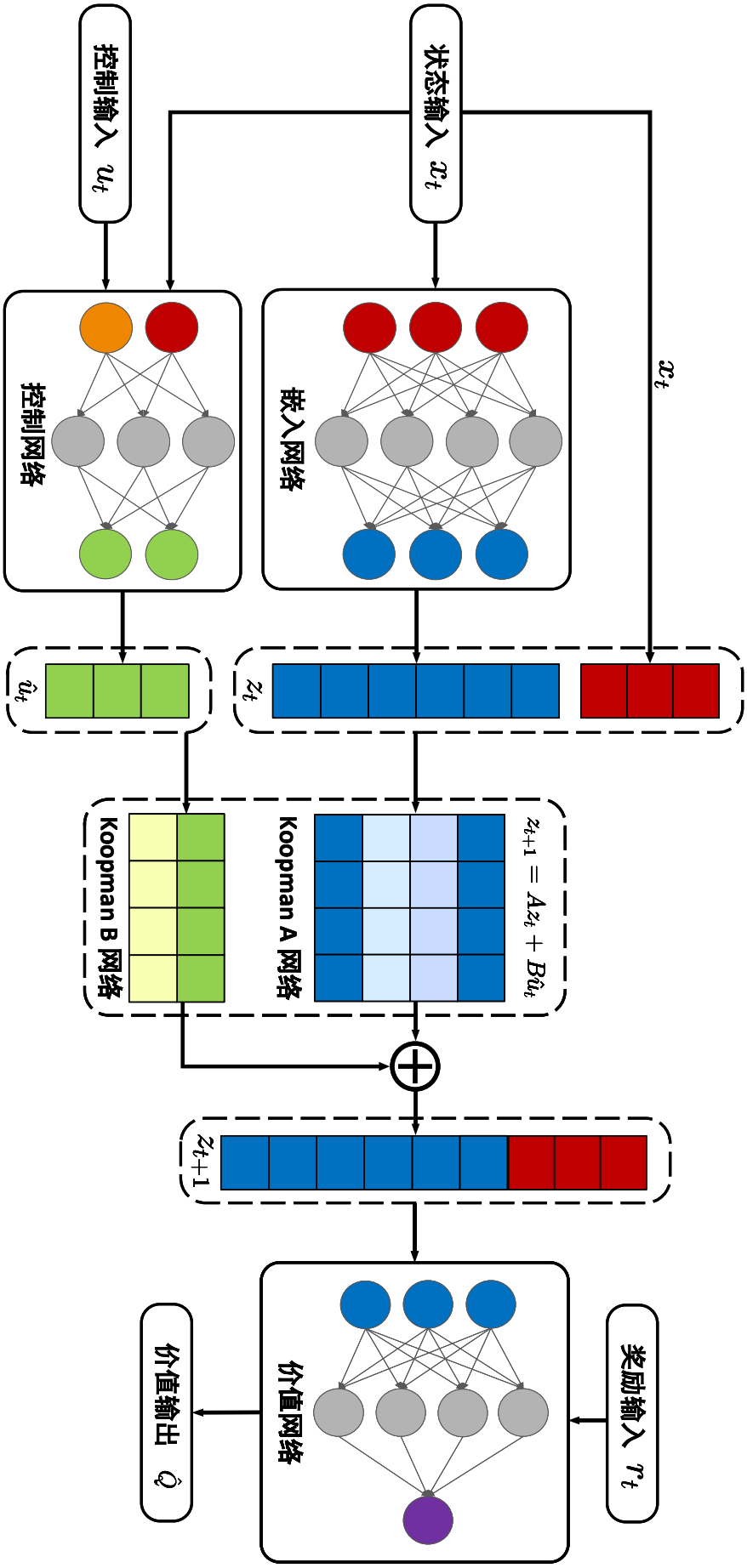

图 A.1　基于非线性形式辅助网络的深度 Koopman 价值网络

致　　谢

这里是致谢页

（我是谁？兰朵儿开发者：余航，致谢我，查重时一定会重复的，哈哈，开个玩笑，本科生论文不在查重范围，而且“毕业论文 (设计) 检测内容主要为毕业论文 (设计) 的主体部分”)。

毕业论文（设计）成绩表

导师评语

建议成绩_____签名_____　　指导教师（签字）_____签名_____

答辩委员会意见

答辩委员会负责人（签字）_____

成绩_____　　**学院（盖章）_____**

年　月　日